局所的な次数情報を用いた無向グラフの探索 (理論計算機科学の深化 : 新たな計算世界観を求めて)

AUTHOR(S):

来見田, 裕一; 小野, 廣隆; 定兼, 邦彦; 山下, 雅史

CITATION:

来見田, 裕一 ...[et al]. 局所的な次数情報を用いた無向グラフの探索 (理論計算機科学の深化 : 新たな計算世界観を求めて). 数理解析研究所講究録 2008, 1599: 127-132

ISSUE DATE:

2008-05

URL:

http://hdl.handle.net/2433/81785

数理解析研究所講究録
第1599巻 2008年 127-132

局所的な次数情報を用いた無向グラフの探索

来見田 裕一 * 小野 廣隆 † 定兼 邦彦 † 山下 雅史 †
* 九州大学大学院システム情報科学府
† 九州大学大学院システム情報科学研究院

1 はじめに

本論文では与えられた連結な無向グラフを探索する問題を考える．ここでグラフ探索とはグラフ上を辺に沿って遷移するトークンが，事前にグラフ全体の構造に関する情報を一切持たず，ある開始点から出発し全ての頂点に訪れ，また全ての辺を渡ることをいう．グラフ探索は，ネットワークにおける頂点に格納されたデータの検索や，ネットワークメンテナンスなどといった，未知のネットワークの内部を調べる問題において基本的なタスクとなる．

グラフ探索問題ではトークンの能力やグラフに関する情報の入手可能性などに対して様々な設定のもとで研究が行われてきた．本研究では，頂点が一意にラベル付けされていることと，探索においてトークンは事前に頂点数に関する情報を持たず，また頂点には情報を書き残せないと仮定する．グラフ探索アルゴリズムの性能の評価指標には，一般的に探索ステップ数（探索終了までにトークンが渡る辺の数）とメモリ量（トークンが必要とする記憶域）があるが，本研究ではメモリ量に着目する．

筆者らはこれまでに，トークン近傍の頂点の次数情報を用いるグラフ探索アルゴリズム *forest search* を提案した [3]．forest search では，隣接する頂点間の親子関係がそれらの次数によって決められ，全体としてはグラフの全域森が定義される．トークンはこの仮想的な全域森の構造に従い，深さ優先探索（DFS）に基づいて頂点間を遷移する．局所的な木構造はトークンの近傍頂点の情報を用いて一意に決定できるため，記憶する必要があるのは木々の間を遷移するのに必要な情報のみであり，グラフ構造によっては省メモリな探索が可能となる．[3] では，計算機実験を通して頂点の次数分布がベキ則に従うある種のグラフに対し，探索ステップ数とメモリ量に関して共に効率的な探索が行えることを示した．しかしながら，一般にはグラフの頂点数 n に対して，トークンは $O(n \log n)$ ビットのメモリを必要とする．

本論文では，forest search の枠組みで使用するメモリ量をどこまで減らすことが可能かという理論的な観点から，アルゴリズムの改良を行う．USTCON(無向グラフにおいて与えられた二頂点間にパスが存在するか決定する問題) を解くために [5] で提案されているアルゴリズムを参考に，グラフ全体として見たときには仮想的な木構造を再帰的に構築するようなアルゴリズムを作ることにより，使用するメモリ量を $O(\log^2 n)$ ビットにできることを示す．

本文の構成は以下の通りである．まず，2節において本研究で扱う探索問題のモデルと関連研究について述べる．次に3節で筆者らが以前提案した探索手法である forest search を説明する．4節では forest search のアルゴリズムの変更と，変更後の探索でトークンが必要とするメモリ量について述べる．最後に5節でまとめと今後の課題を示す．

2 準備

グラフ探索問題は未知な環境内をロボットが探索する問題のひとつに位置付けられ，これまでに様々なモデル設定の下で研究が行われてきた．本研究で扱うモデルでは特に以下のことを仮定する．

1. 無向グラフ $G = (V, E)$ はある分散システムの通信ネットワークをモデル化したものであり，頂点と辺はそれぞれサイトと通信リンクを表す．この通信ネットワークはロジカルである．

2. 集中型モデルではない．すなわち，トークンは訪れたことの無い頂点に関する情報を入手でき

ない.

3. 各頂点は一意のラベル (ID) を持つ. ID は整数で表されている.
4. 各頂点はそこへ接続する辺を識別できるが, その辺によってどの頂点と接続しているかを認識できない.
5. トークンは探索前に G の構造や頂点数 $n = |V|$ について一切の情報を持たない.
6. トークンは頂点に情報を書き残すことができない.
7. トークンはいくつかの頂点の ID を記憶することができ, メモリに格納されている頂点へは直接移動 (ジャンプ) することができる.

仮定 2 について, 集中型モデルとは G がランダムアクセス可能な行列として表現されている場合を指す. 集中型モデルであれば, USTCON が $O(\log n)$ ビットで解けることを証明した論文 [4] によって, 無向グラフを $O(\log n)$ ビットで探索可能なアルゴリズムが存在することが示されている. また仮定 3 について, 各頂点が ID を持たない無向グラフであれば, $\Theta(D \log d)$ ビットで探索可能であることが証明されている [2]. ここで, D, d はそれぞれグラフの直径と最大次数を表す.

3 forest search

実世界に存在する多くのネットワークでは次数の分布がベキ則に従うことが知られている [1]. すなわち, ある頂点が次数 d を持つ確率 $P(d)$ は定数 γ を用いて, $P(d) \propto d^{-\gamma}$ と書ける. このようなネットワークはスケールフリーネットワークとも呼ばれる. 筆者らは以前, 探索の対象をスケールフリーなグラフに限定することでグラフの規模が非常に大きい場合でも効率の良い探索を行うアルゴリズム forest search を提案した [3]. forest search ではスケールフリーという構造の特徴を活かすため, トークンの近傍頂点の次数情報を用いて探索を行う. 以下でその方法を述べる.

グラフの各頂点 u に対して, $N^k(u)$ を u からの距離が高々 k である頂点の集合とする. 各頂点 u に対して, $N^1(u)$ の中から最大次数の頂点をその親 $\pi(u)$ に選ぶ. 最大次数の頂点が複数ある場合にはその中で ID が最小の頂点を選ぶ. 与えられたグラフ G の全頂点と各頂点 u に対して辺 $\{u, \pi(u)\}$ からなるグラフを F とすると, F には長さ 2 以上の閉路は存在しないこと, つまり森であることが示せる. 自己ループのある頂点はそれが木の根であることを意味する.

図 1: forest search

Step 0:	root list と cross stack を空にした後, 開始頂点へ.
Step 1:	現在の頂点から根 u まで登る. u が root list にあるか調べ, あれば cross stack から横断辺を pop してその遷移元頂点へジャンプし, Step 3 へ.
Step 2:	現在の頂点 (根) を root list へ追加し, 仮想木を DFS で探索.
Step 3:	仮想木の内部を DFS と同様に辿りながら次の横断辺を探す. あればそれを cross stack に push した後渡り, Step 1 へ. なければ cross stack から横断辺を pop してその遷移元頂点へジャンプし, Step 3 へ. (cross stack を pop するとき空ならば, 連結成分の全頂点に最低一度は訪れたことを意味する.)

図 1 にアルゴリズムの骨子を示す. forest search はトークン近傍の次数情報を用いて F の局所構造を求め, それに沿ってトークンを遷移させることで G が森であるかのように探索する. 探索の概要としては目標頂点が見つかるまで新しい木を探し, その内部を DFS によって探索することを繰り返す. 親の定義のおかげで, 木の内部を DFS で探索する際に経路を記憶する必要は無い. forest search によって木は暗に定義されるので, これを仮想木と呼ぶ. 仮想木を識別し, また木の間を行き来するため, トークンの記憶域としてはそれまでに発見した木の根の ID を保持する *root list* と, 木に含まれない辺 (横断辺) を記憶する *cross stack* が必要である. cross stack の一要素は遷移元, 遷移先頂点の ID の順序対からなる有

向辺である.

実際にはこのアルゴリズムは，root list と cross stack の情報を用いることで各仮想木を擬似的な頂点としたときの多重グラフにおいても DFS を実行するが，木の根まで行かなければその木が既に探索済みの木か判別できないため，現在の木に“隣接する”全ての木に訪れる必要がある．そのため通常の DFS とは異なり，最悪の場合 G を $O(t \log n)$ ビットのメモリ量を用いて $O(n+|E|\cdot h)$ ステップで探索する．ここで t, h はそれぞれ F における連結成分（木）の数と最大の深さを表す.

計算機実験を通じて forest search はスケールフリーグラフにおいてはメモリ量に関して効率的な探索が可能であるという結果が得られた．しかし一般のグラフにおいては $t = O(n)$ であるため，トークンが使用するメモリ量は $O(n \log n)$ ビットになる.

4 再帰的な forest search

基本的なアイデアとしては，G の仮想森を再帰的に構築し，各再帰のレベルにおける仮想木で DFS を実行するというものである．本節では，まず再帰的 forest search が定義する階層的な仮想森の構造について述べた後，その構造に従ってトークンを遷移させる方法について示す.

与えられたグラフに対する仮想構造の再帰的な構築という概念は，USTCON を解くためのアルゴリズムにしばしば用いられる *hook and contract* というアプローチに見られる．近年の例としては [5] が挙げられる．このアプローチは二つのフェーズから成り立つ．ここで，$G_c = (V_c, E_c)$ を与えられた（連結とは限らない）無向グラフとし，$s, t \in V_c$ を二つの異なる頂点とする.

- hooking: 各頂点 $u \in V_c$ に対して，u の隣接頂点の中からひとつを選ぶ（*hook*）．どの頂点を選ぶかは，アプローチの実装に依存する．hook の向きを無視すると，G_c の全頂点は hooking によってクラスタに分割される.
- contract: hooking によって作られた各クラスタを新たなグラフ G_c' の頂点へ対応させる．全ての異なる二頂点に対して，G_c において対応するクラスタの間にパスが存在するときに限り，その二頂点間に辺を張る.

これらのフェーズを G_c における各連結成分が一頂点に対応付けられるまで再帰的に適応させることによって，G_c において s, t 間にパスが存在するかどうかが自明になる.

再帰的 forest search では，各仮想木を hooking フェーズにおけるクラスタと見なす．contract フェーズに関しては，トークンはグラフ構造を変更することは不可能であり，またメモリ量を削減することを考えるため，新たなグラフを記憶することもできない．そのため，構造を保持する代わりに，トークンに仮想木の内部を遷移させることによって contract のプロセスをアルゴリズムに組み込む．トークンの遷移方法の詳細と必要なメモリ量については後述する.

仮想森の再帰的な構築は階層的なグラフの全域森を暗に定義する．以後，各階層（再帰）と階層（再帰）の深さをそれぞれ**レイヤ**，**レイヤの数**と表記する．このとき，頂点がラベル付けされた任意の無向連結グラフ $G = (V, E)$ に対して，レイヤ数は $O(\log n)$ になることが示せる．ただし，$n = |V|$ である．始めに再帰的ではない forest search について考える．G に対して，forest search が定義するグラフ F の各木をそれぞれ一点に縮約し，縮約後の頂点 ID に縮約前の木の根の ID を割り当てて得られるグラフを $G' = (V', E')$ と表す.

補題 1 $|V'| \leq \frac{|V|}{2}$.

証明. forest search の親選びを適用すると木の根となる頂点集合を V_r とする．親の決め方により，G で隣接するどの 2 頂点 $u, v \in V$ についても u, v が共に V_r に入ることはないので $V_r \subsetneq V$ である．また，木の根となる頂点 $r \in V_r$ に隣接する頂点 w は $V \setminus V_r$ に属する．G におけるこのような頂点 w を減少点と呼び，減少点全体の集合を V_i とおく．$V \setminus V_r$ の中にはどの $r \in V_r$ とも隣接していない頂点がありうるため，$V_i \subseteq V \setminus V_r$ である.

さらに，各 $r \in V_r$ と r に隣接する減少点 $w \in \mathrm{N}(r)$ の頂点間において，次数関係 $\deg(r) \geq \deg(w)$ が成り立つ．ここで $\deg'(w) := |\{u | \{w, u\} \in E \wedge u \in V_r\}|$ とする．各減少点 w はその隣接する $r \in V_r$ によって“等分”されていると考えて，r が持つ減少点の数を

$I(r)$ とすると，

$$\begin{aligned} I(r) &= \sum_{w\in N(r)} \frac{1}{\deg'(w)} \geq \sum_{w\in N(r)} \frac{1}{\deg(w)} \\ &\geq \sum_{w\in N(r)} \frac{1}{\deg(r)} \\ &= \deg(r)\cdot\frac{1}{\deg(r)} = 1 \end{aligned}$$

となる．この式はひとつの r あたり排他的に少なくとも 1 個の減少点が存在すると解釈することができる．つまり，$\sum_{r\in V_r} I(r) \geq |V_r|$ となる．また，$|V_i| = \sum_{r\in V_r} I(r)$ である．

したがってグラフの頂点数に関して，

$$\begin{aligned} |V| &= |V_r| + |V \setminus V_r| \\ &\geq |V_r| + |V_i| \\ &= |V_r| + \sum_{r\in V_r} I(r) \\ &\geq 2|V_r| \\ &= 2|V'|. \end{aligned}$$

□

補題 1 より，次が成り立つ．

定理 1 再帰的 *forest search* において，レイヤ数 (再帰の深さ) は高々 $\lceil \log n \rceil$．

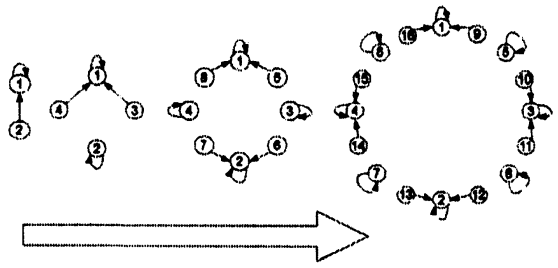
実際にレイヤ数が $\lceil \log n \rceil$ になるグラフは例えば図 2 ような進化グラフのモデルで構成できる．構成したいグラフはサイクルグラフだが，説明ではそのグラフに forest search での親選びを適用した森をベースに記述している．グラフの成長過程は図 3 のようになる．

以降では，上記の階層的な仮想構造においてトークンを遷移させる方法について述べる．アルゴリズムには，擬似頂点と見なした各仮想木がその親を選ぶことによって，“上のレベルの”仮想木を構築するという再帰的な過程が組み込まれている．この過程はグラフに存在する擬似頂点がひとつになるまで繰り返される．トークンが仮想構造に従って遷移することにより，forest search から root list と cross stack を除去できる．各レイヤにおいて必要となる情報を除くと，トークンが記憶しなければならない情報は

図 2: 進化グラフのモデル

Step 0:	二頂点 (ID: 1,2) から成るひとつの根付き木を用意する．
Step 1:	グラフの各頂点 u に対して，u が根でないなら根にする (親から切り離して自己ループを付ける)．u が根なら新たに二個の頂点を u の子としてグラフに追加する．
Step 2:	Step 1 で新たに追加した頂点の ID を適当な順に割り当てて行く．Step 1 へ戻る．

図 3: 進化グラフの成長過程



現在トークンがいるレイヤのレベルと頂点の ID だけである．

以後，レイヤ *level* においてトークンが存在する仮想木に対して，トークンが根まで登る，DFS を実行するという動作をそれぞれ climb(*level*), DFS(*level*) と表記する．同様に，レイヤ *level* において現在トークンがいる擬似頂点に隣接する擬似頂点の集合を $N(level)$ と表す．擬似頂点の ID は対応する仮想木の根の ID である．各レイヤにおいて，擬似頂点の次数，すなわち対応する仮想木に接続する cross edge の数，および隣接擬似頂点へ遷移するための辺は DFS(*level*) を実行することによって得る．したがって実際には，再帰過程は線形に進められるわけではなく，再帰レベル間をジグザグに進む．つまり，トークンは climb(*level*) と DFS(*level*) 実行中に，*level* 以下の異なるレイヤにおける仮想木の間を何度も行き来しなければならない．

アルゴリズムの最も外側の記述を図 4 に示す．deg(*level*) はレイヤ *level* において現在トークンがいる擬似頂点の次数 (多重辺は含めない．多重辺の

図 4: 再帰的 forest search

Step 0:	$level := 0$
Step 1:	climb($level$)
Step 2:	if (deg($level$) = 0) halt
Step 3:	DFS($level$)
Step 4:	$level := level + 1$
Step 5:	goto Step 1

検出方法については後述）である．図 4 では Step 2 としてアルゴリズムの終了条件が示されているが，deg($level$) は実際には Step 1 における climb($level$) によって求められる．

前述のようにレイヤ $level$ における仮想木の構造は $level-1$ 以下の再帰過程によって決定されるが，説明の簡単のため，以下ではひとつのレイヤにおける動作に着目する．climb($level$) ではトークンが仮想木の中をただ登るだけであり，DFS($level$) のわずかな変更で実装可能であるため，ここでは DFS($level$) のみを解説する．各レイヤに対して必要になる記憶域については，仮に全てのレイヤにおける仮想構造が単純グラフであれば，トークンがレイヤ $level$ で近傍の親子関係を決定するのに必要であり，記憶しなければならない情報は，

- 現在トークンがいる擬似頂点 u の ID と次数 deg($level$),
- トークンがある隣接擬似頂点を訪問するとき，その ID と次数,
- $N(level)$ と u の中で，最大次数を持つ隣接擬似頂点の ID と次数.

しかしながら一般には，階層的な仮想森においてトークンは多重グラフを探索しなければならない．この場合，DFS($level$) の実行中に探索経路に閉路が生じるのを避けるため，トークンが仮想木の中で擬似的な子に下りるときには常にその擬似頂点へ接続する多重辺の一本目を渡るようにする．この処置のため，トークンが擬似頂点へ渡るときには更に以下の情報が必要となる．

- トークンが渡る辺の遷移元（擬似ではなくオリジナルの）頂点の ID,

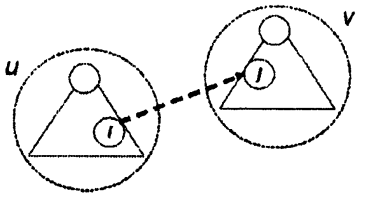


図 5: 擬似頂点 u, v と内部頂点 i, j, ただし $\{i, j\} \in E$

- トークンが渡る辺の遷移先擬似頂点の ID.

例えば，図 5 では，遷移元頂点および遷移先擬似頂点はそれぞれ i, v として示されている．トークンはこれらの ID の順序対を二組必要とする．一組目は DFS($level$) 実行中に擬似頂点へ訪れるときにトークンが渡る有向辺 e を保持するためであり，二組目は e と並行な一本目の辺を探索するためのものである．レイヤ $level$ において，一組目，二組目の対をそれぞれ $(s_1, d_1), (s_2, d_2)$ とし，u を起点となる擬似頂点とする．トークンはこれらの情報を用いて以下のようにレイヤ $level$ の仮想木を探索する．まず，$e = (s_1, d_1)$ を渡ることで仮想木の構造から外れたときには，トークンは s_1 へジャンプし，次の隣接擬似頂点を探索し訪れる．構造から外れたかどうかは，近傍の親子関係によって検出可能である．

仮想木を下りる場合（図 6 の下部参照）: $e = (s_1, d_1)$ を渡る際に記憶し，d_1 が u の擬似頂点であると判明した後，u（に対応する仮想木の根）へジャンプする．DFS($level-1$) によって $level$ において u に接続する辺を最初から走査し，各辺に対して s_2 を記憶して渡り，climb($level-1$) を実行，対 (s_2, d_2) の二番目の要素となる d_2 を覚えて u へジャンプするという動作を $d_1 = d_2$ となるまで繰り返す．$d_1 = d_2$ かつ $s_1 = s_2$ であれば，(s_1, d_1) で表される辺は u と d_1 を接続する多重辺の一本目の辺であるため，擬似頂点 d_1 から更に深く探索する．そうでなければ辺 (s_1, d_1) は一本目の辺ではないため，s_1 へジャンプして戻り u の次の子へ接続する辺を探索する．

仮想木を登る場合（図 6 の上部参照）: トークンが木を登るときに渡る辺 e' 自体を気にする必要はないが，トークンは次の子へ接続する一本目の辺を見つけ，それを渡らなければならないため，e' の逆辺 e と並行な一本目の辺を探索する必要がある．現在トークンがいる擬似頂点 d_1 の ID を記憶した後，その親へ接

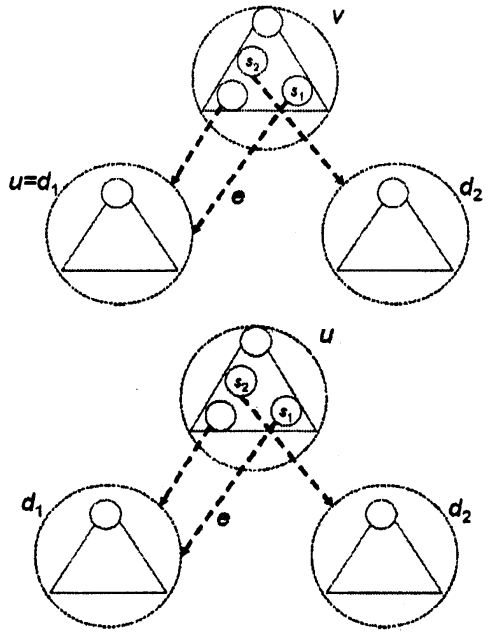


図 6: 仮想木での上り（上部）下り（下部）

続する辺 e' を渡り，climb($level-1$) を実行して u の親 v の内部にある根まで登る．DFS($level-1$) によって $level$ において v に接続する辺を最初から走査し，各辺に対して s_2 を記憶して渡り，climb($level-1$) を実行，対 (s_2, d_2) の二番目の要素となる d_2 を覚えて v へジャンプして戻るという動作を $d_1 = d_2$ となるまで繰り返す．$d_1 = d_2$ となったとき，(s_2, d_2) で表される辺は e と並行な一本目の辺であるため，s_2 へジャンプして戻る．s_2 から辺の探索を再開することで，トークンは次に渡るべき d_1 の子へ接続する一本目の辺を逃すことはない．

トークンはレイヤ $level$ において上記のデータを一段階の親子関係分のみ保持しておけばよく，それらを用いることで DFS($level$) の実行に必要な情報をその場で求めることができる．したがって，レイヤごとに記憶しなければならない ID や次数の数は定数であり，トークンは各レイヤに対して $O(\log n)$ ビットのメモリ量を必要とする．レイヤ数は高々 $\lceil \log n \rceil$ であり，大域的に保持する情報は $O(\log n)$ ビットで済むため，再帰的 forest search は n 頂点の任意の連結グラフを $O(\log^2 n)$ ビットのメモリ量で探索する．

5 まとめと今後の課題

集中型ではないモデルにおいて，forest search に基づき，頂点がラベル付けされた無向グラフを $O(\log^2 n)$ ビットで探索する手法を提案した．

今後の課題としては，より少ないメモリ量での探索可能性や他の探索モデル・USTCON との関係性の解明が考えられる．

参考文献

[1] R. Albert and A.-L. Barabási, Statistical Mechanics of Complex Networks, Reviews Of Modern Physics, Vol.74, pp. 47–97, 2002.

[2] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc and D. Peleg, Graph Exploration by a Finite Automaton, Theoretical Computer Science 345, pp. 246–257, 2005.

[3] Y. Kurumida, H. Ono, K. Sadakane and M. Yamashita, Forest Search: A Paradigm for Faster Exploration of Scale-Free Networks, Proc. of ISPA 2006, pp. 39–50, 2006.

[4] O. Reingold, Undirected ST-connectivity in log-space, Proc. of STOC 2005, pp. 376–385, 2005.

[5] V. Trifonov, An $O(\log n \log\log n)$ space algorithm for undirected st-connectivity, Proc. of STOC 2005, pp. 626–633, 2005.